

We kunnen zelden alle gewenste data bewaren in aparte variabelen (vb wanneer je een programma moet maken dat het gemiddelde berekent van 100 getallen). Aangezien het aantal variabelen dat moet worden bijgehouden afhankelijk is van verschillende factoren is het nodig om constructies te gebruiken die deze variërende factor kunnen opvangen. Arrays zijn de meest primitieve vorm van massa containers die we in C# zullen gebruiken. Collections zijn hier een uitbreiding op en bieden extra functionaliteit die het werken met deze gegevensopslag makkelijker maakt. Arrays zijn echter de basis van collecties en vele eigenschappen (zoals het indexing mechanisme) vinden we terug in de geavanceerde Collections.

De “tekortkomingen” van Arrays werden weggewerkt in de collections objecten (ArrayList) en nog meer in de generische vorm hiervan (vb de populaire generische List<Type>). Deze collections laten ons toe om gemakkelijk een aantal objecten bij te houden zonder, tijdens de ontwikkeling van ons programma, de grootte te moeten aangeven. Collections zijn dus dynamische Arrays. Verder gaan collections ons ook verschillende functies aanbieden die het werken met deze structuren gemakkelijker zal maken.

Oefening 1

Maak een programma om een wagenpark te beheren. In het menu worden de volgende opties getoond (menu opties 4 en 5 zijn extra en dus enkel toe te voegen indien gewenst):

1. Voeg een wagen toe => voegt een wagen toe aan het wagenpark. Indien geen nummerplaat werd ingegeven zal het onmogelijk zijn de wagen toe te voegen.
2. Geef wagenpark overzicht => Toont een lijst van alle wagens die aan het wagenpark werden toegevoegd
3. Voer expertise uit => de schade aan alle wagens in het wagenpark zal worden bepaald door een extern expertisebureau
4. Voeg een garage toe => voegt een garage toe aan de collectie van garages
5. Repareer wagen => zal een gekozen wagen (op basis van nummerplaat) repareren. Op basis van het merk van de gekozen wagen zal een overeenkomstige garage gezocht worden. Indien geen overeenkomst in merk zal de wagen niet kunnen gerepareerd worden.
6. Stop => stopt het programma

We onderscheiden de volgende klassen (gebruik ook deze volgorde om je programma op te bouwen!):

Wagen <klasse>

Properties

- Nummerplaat <string>: De nummerplaat van de wagen
- Merk <string>: Het merk van de wagen
- Schade <double>: de schade aan de wagen

Constructoren

- Default constructor: initialisatie data members op default waarden
- Constructor met waarde voor nummerplaat en merk. Voor een nieuwe wagen is de schade steeds 0

WagenPark <klasse>

Properties

- Wagens <List<Wagen>>: de collectie van wagens

Constructoren

- Default constructor: initialisatie van Wagens!

Functies

- SchrijfWagenIn
 - Return <bool>
 - Parameters
 - wagen <Wagen>: de in te schrijven wagen

- Info: Indien de wagen een blanco nummerplaat heeft zal deze niet aan de Wagen collectie worden toegevoegd (return false).
- GeefOverzicht
 - Return <string>
 - Parameters: geen
 - Info: geeft een omschrijving terug van het wagenpark (nummerplaat / Merk en schade per wagen)
- ZoekWagen
 - Return <Wagen>
 - Parameters
 - nummerplaat <string>: de nummerplaat van de te zoeken wagen
 - Info: Indien de wagen wordt gevonden zal deze worden teruggegeven, null in het andere geval.

ExpertiseBureau <klasse>

Functies

- VoerExpertiseUit
 - Return <void>
 - Parameters
 - wagens <List<Wagen>>: de collectie van de na te kijken wagens
 - Info: de functie zal een waarde toekennen aan de schade eigenschap van alle wagens in de ontvangen collectie (`double schade = random.NextDouble() * 10000;`). Indien de wagen reeds een schade heeft zal de nieuwe schade hierbij opgeteld worden.

Garage <klasse> [EXTRA]

Properties

- Naam <string>: De naam van de garage. Niet aanpasbaar door een andere klasse
- Merk <string>: Het merk van de wagen. Niet aanpasbaar door een andere klasse

Constructoren

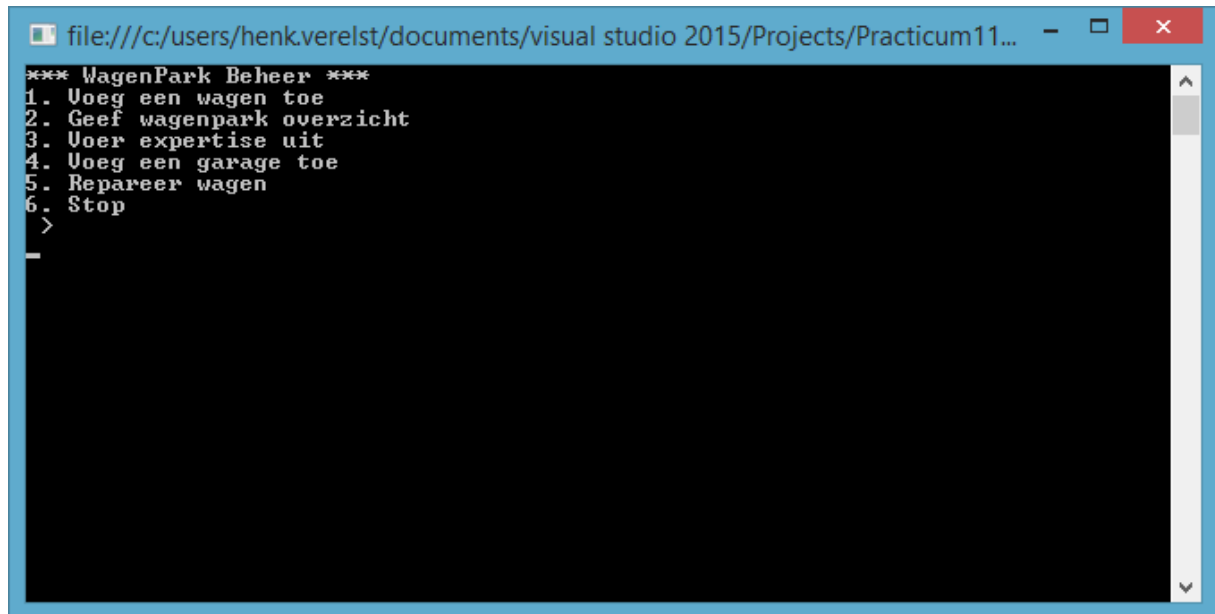
- Default constructor: initialisatie data members op default waarden
- Constructor met waarde voor naam en merk.

Functies

- RepareerWagen
 - Return: double (de totale kost van de herstelling)
 - Parameter:
 - wagen <Wagen>: de te repareren wagen
 - Info: Zal de wagen herstellen = schade van object(en) op 0 zetten en de kost van deze herstelling teruggeven

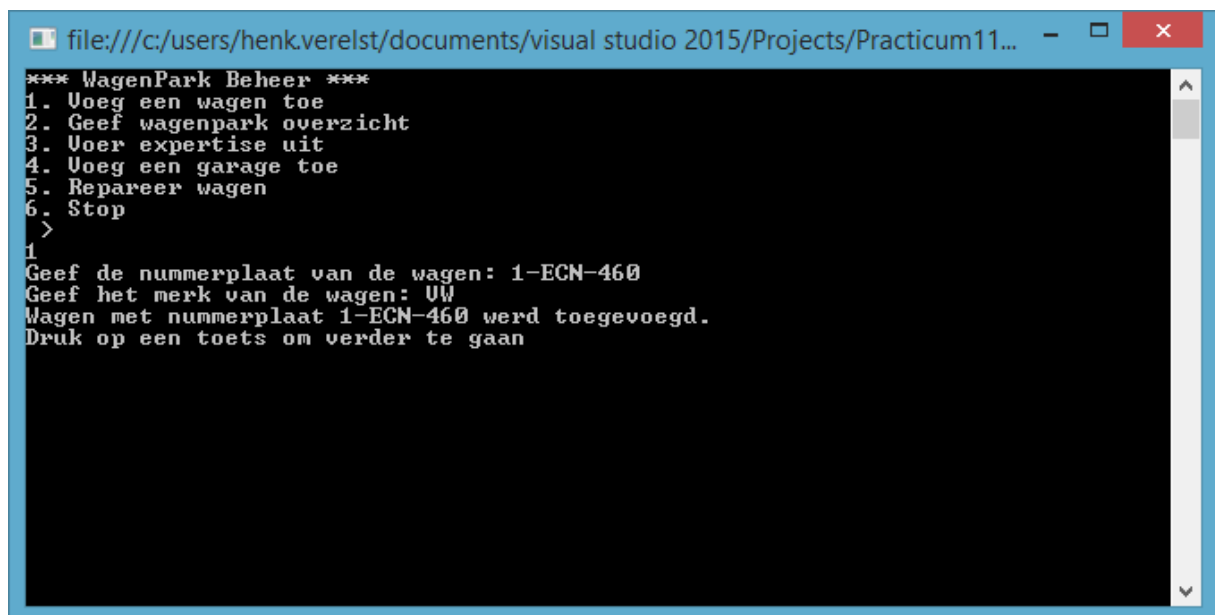
Hieronder een overzicht de verschillende schermen:

Hoofdmenu



```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>
```

Schrijf wagen in



```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>  
1  
Geef de nummerplaat van de wagen: 1-ECN-460  
Geef het merk van de wagen: UW  
Wagen met nummerplaat 1-ECN-460 werd toegevoegd.  
Druk op een toets om verder te gaan
```

Overzicht wagenpark

```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11... - [X]
*** WagenPark Beheer ***
1. Voeg een wagen toe
2. Geef wagenpark overzicht
3. Voer expertise uit
4. Voeg een garage toe
5. Repareer wagen
6. Stop
>
2
*** Wagenpark Overzicht ***
- 1-ECN-460 /Merk: UW / Schade: 0
- 1-EYP-005 /Merk: Audi / Schade: 0
Druk op een toets om verder te gaan
-
```

Voer expertise uit

```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11... - [X]
*** WagenPark Beheer ***
1. Voeg een wagen toe
2. Geef wagenpark overzicht
3. Voer expertise uit
4. Voeg een garage toe
5. Repareer wagen
6. Stop
>
3
De expertise op het wagenpark werd uitgevoerd
Druk op een toets om verder te gaan
-
```

Voeg een garage toe [extra]

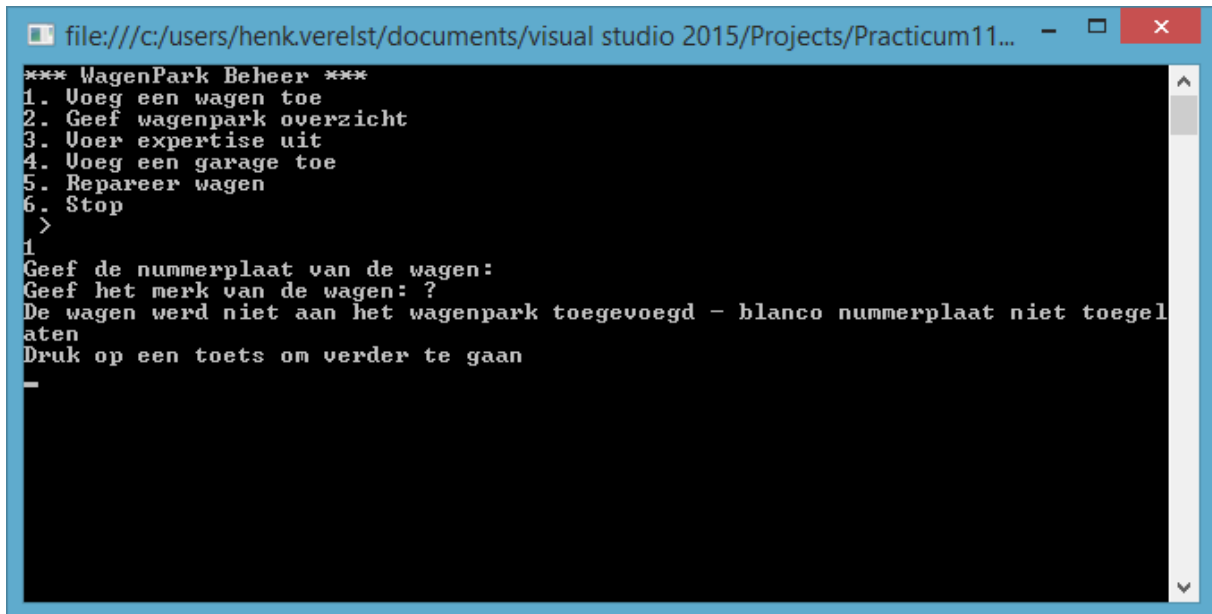
```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>  
4  
Geef de naam van de garage: UCLL UW  
Geef het merk van de garage: UW  
Garage UCLL UW werd toegevoegd.  
Druk op een toets om verder te gaan
```

Repareer wagen [extra]

```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>  
5  
Geef de nummerplaat in van de te repareren wagen: 1-ECN-460  
De wagen werd gerepareerd door garage UCLL UW Totale kost: 2151,01  
Druk op een toets om verder te gaan
```

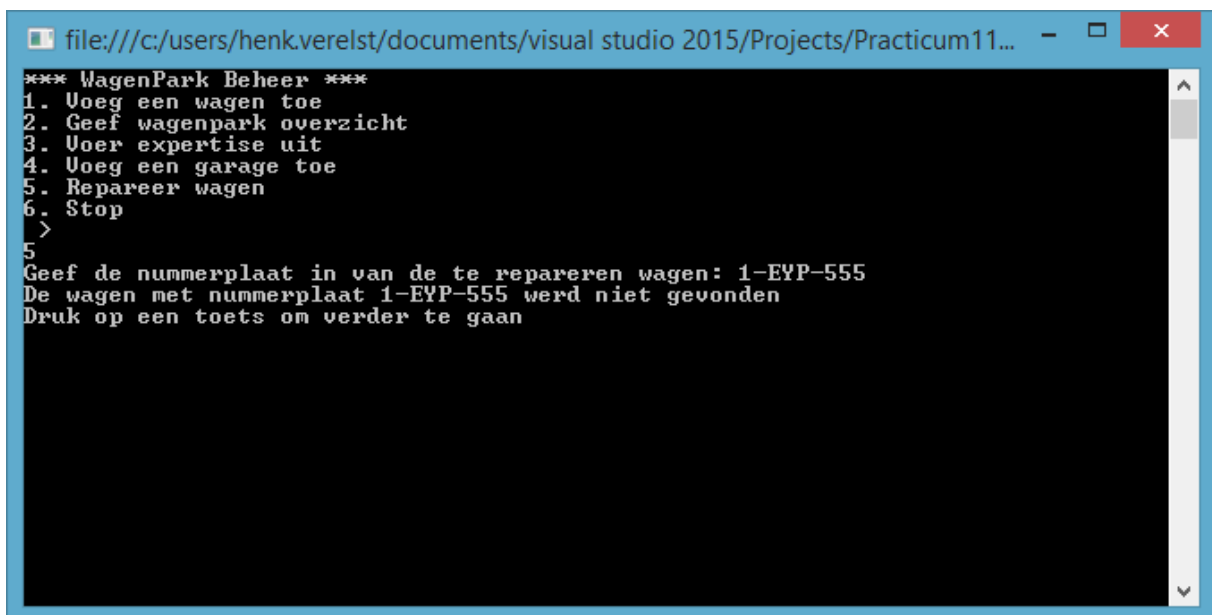
Mogelijke fouten:

Blanco nummerplaat:



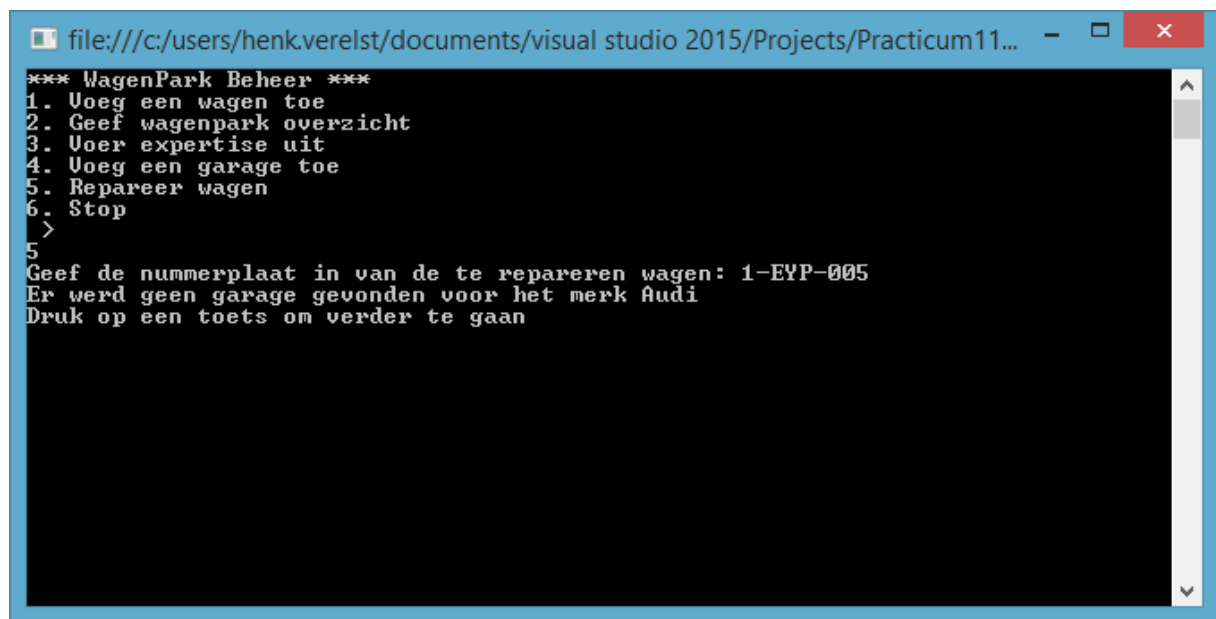
```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>  
1  
Geef de nummerplaat van de wagen:  
Geef het merk van de wagen: ?  
De wagen werd niet aan het wagenpark toegevoegd - blanco nummerplaat niet toegelaten  
Druk op een toets om verder te gaan  
-
```

Nummerplaat wagen niet gekend



```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>  
5  
Geef de nummerplaat in van de te repareren wagen: 1-EYP-555  
De wagen met nummerplaat 1-EYP-555 werd niet gevonden  
Druk op een toets om verder te gaan
```

Geen garage voor merk wagen



```
file:///c:/users/henk.verelst/documents/visual studio 2015/Projects/Practicum11...  
*** WagenPark Beheer ***  
1. Voeg een wagen toe  
2. Geef wagenpark overzicht  
3. Voer expertise uit  
4. Voeg een garage toe  
5. Repareer wagen  
6. Stop  
>  
5  
Geef de nummerplaat in van de te repareren wagen: 1-EYP-005  
Er werd geen garage gevonden voor het merk Audi  
Druk op een toets om verder te gaan
```